C++ Implementation of a Cryptosystem

Marcus Trinder
University of Manchester
3^{rd} Year Lab Report

***Abstract*— A class hierarchy to support cryptographic algorithms is constructed using C++ object orientated programming. System objects are created which enable encryption and decryption of data using a specific cryptosystem. Implementation of the RSA algorithm is shown with user input and also external text files.**

I. INTRODUCTION

The rise of the computer age in the mid-twentieth century has meant increased communication via digital data. Some communication is required to be protected; from governmental procedures to personal financial details. The topic of protecting, or encrypting, data is cryptography. A cryptosystem is then a complete set of algorithms required to implement security of data. In modern times, these systems involve growing complexities in order to withstand the coinciding threat of data penetration. One of the earliest cryptosystems is the Rivest–Shamir–Adleman (RSA) public key system, which is presented in this report.

II. MATHEMATICS

One aspect of mathematics that the reader may not be familiar with and that is used heavily in cryptography is modular arithmetic. The most common analogy of modular arithmetic is that of analogue and digital clocks. Usually analogue clocks use a 12-hour system and digital clocks use a 24-hour system. Converting a 'digital time', 16:00 into a 'analogue time, 'four o'clock' is easy, as we all know to subtract twelve and the remainder is the corresponding time. What you may not be familiar is that this can be written as,

$$16 \ mod \ 12, \tag{1}$$

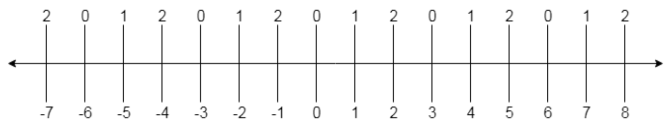


Fig. 1. Visualization of modulo arithmetic. Here you can see than in mod 3 there are 3 possible values for each number. 8 is congruent to 2 and -5 is congruent to 1 in mod 3.

where *mod N* is the modulo operator with modulus N, in this case $N = 12$. The statement in Equation 1 says that the integer 4 is congruent to the integer 16 in modulo 12. Generally, for a given *mod N* operating on an integer, *a* there is a set,

$$Z = \{0, 1, ..., N - 1\}, \tag{2}$$

of possible values of remainders. Then an integer *b* is congruent with *a* if their corresponding values in *mod N* are equal. See Fig 1 for a visual illustration of this.

In C++, the modulo operator is defined by the $\%$ symbol and the equivalence of $a(modN)$ is then $a\%N$.

III. CODE DESIGN

In this project, a program is designed around a class hierarchy allowing for multiple cryptographic algorithms to be initiated. Featured are pure virtual classes *algo* and *assymAlgo*, representing interfaces for algorithm types. These virtual classes contain virtual member functions that are shared across the many available algorithms in existence. Such member functions are the *encypt()* and *decrypt()* functions that implement the encryption and decryption of data respectively. Derived classes, such as the *RSA* class implemented in this program, contain the relative parameters required

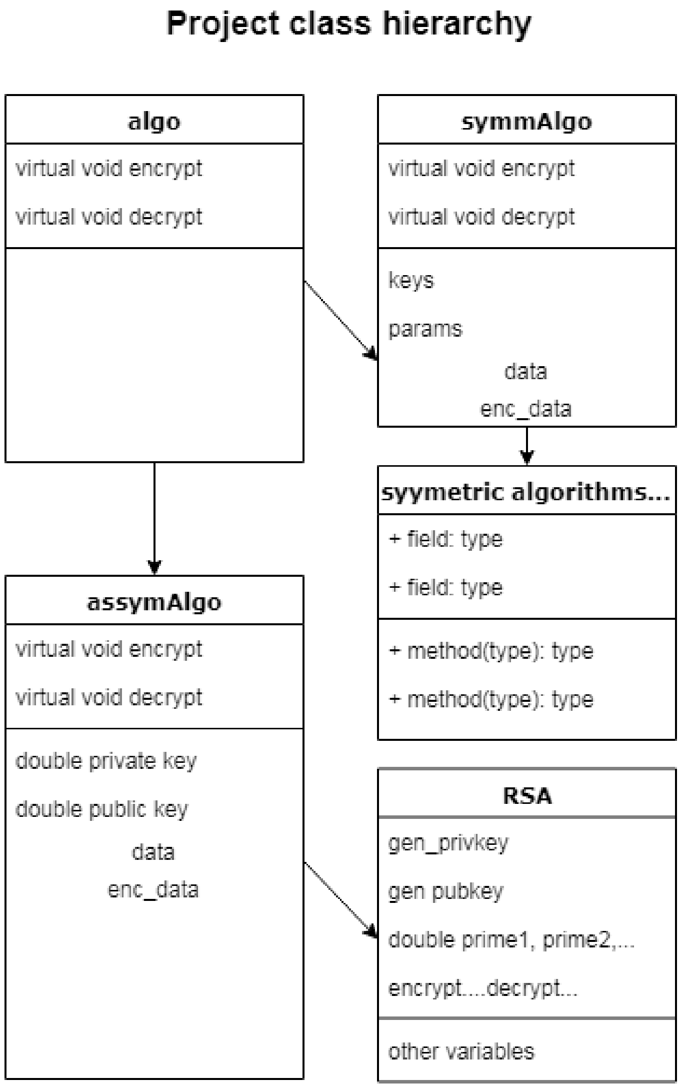
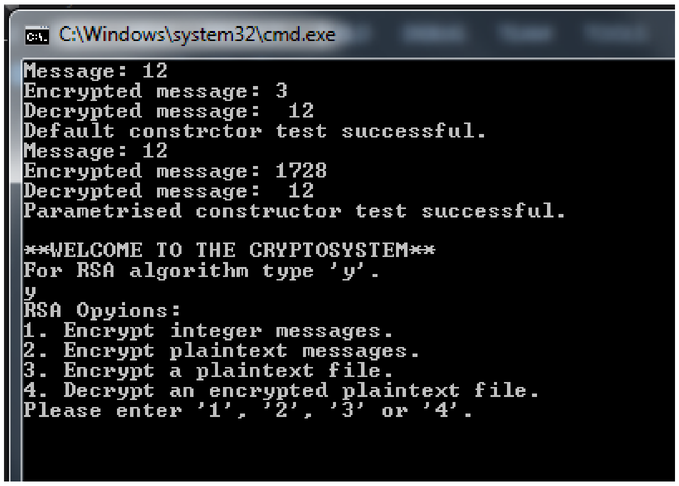


Fig. 2. A class hierarchy diagram of the program consisting of virtual classes algo and assyAlgo, and derived class RSA.

by a specific algorithm. This hierarchy can be visualized in Fig 2. The classes are contained inside the separate header file *Project.h* and member functions are defined inside the file *Project.cpp*. The file *main.cpp* holds the programs main function and implements a chosen cryptosystem on data input by the user. The main function allows for data to be input via direct input of a specified number of integers, a plain text message or an external plain text file using a file stream. When run, the program faces the user with an option list (see Fig 3) where they can select what data to encrypt.


```
C:\Windows\system32\cmd.exe
Message: 12
Encrypted message: 3
Decrypted message:  12
Default constrctor test successful.
Message: 12
Encrypted message: 1728
Decrypted message:  12
Parametrised constructor test successful.

**WELCOME TO THE CRYPTOSYSTEM**
For RSA algorithm type 'y'.
y
RSA Opyions:
1. Encrypt integer messages.
2. Encrypt plaintext messages.
3. Encrypt a plaintext file.
4. Decrypt an encrypted plaintext file.
Please enter '1', '2', '3' or '4'.
```

Fig. 3. Console output of program when run. User inputs 1, 2, 3 or 4. Two tests of the class constructors are ran.

IV. RSA CRYPTOSYSTEM

The RSA cryptosystem is an asymmetric cryptography routine (also known as public key cryptography) as apposed with a symmetric routine that involves a shared key. RSA systems have been used for many different applications in multiple industries historically, but in modern times it has been replaced with more robust techniques [2]. It is used in this project for its simplicity. In the RSA routine, a communicator Alice chooses two prime numbers p and q and calculates $N = pq$. Alice must choose an encryption exponent $e \in N$ which is co-prime to the Euler totient function $\phi = (p-1)(q-1)$. She must then compute the decryption exponent $d \in N$ such that,

$$ed \equiv 1(mod\ \phi). \tag{3}$$

A plain text message, M can be converted to ciphertext, c using,

$$c = M^e(mod\ N). \tag{4}$$

Alice shares her public key (N, e) to Bob who can then decrypt the ciphertext message into the origional message by,

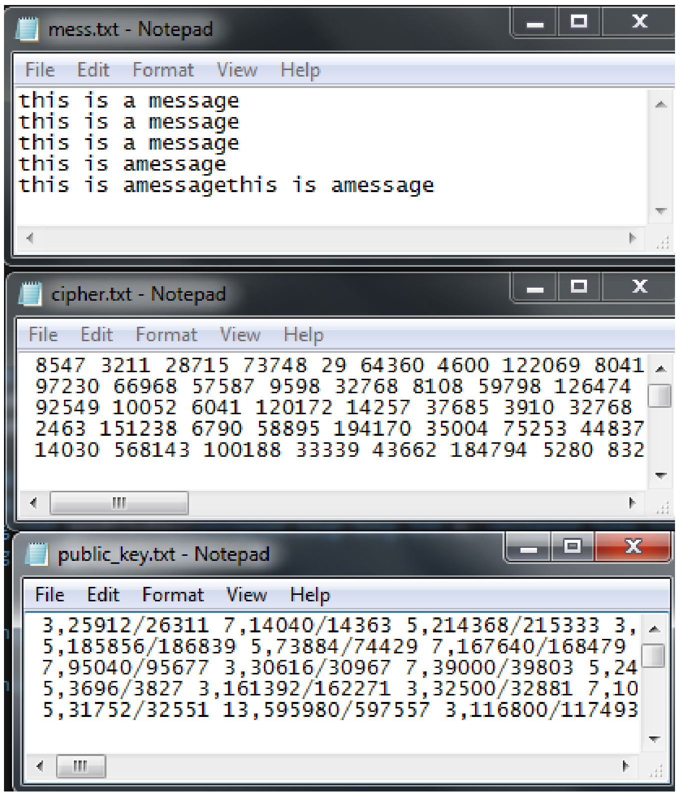
$$M = c^d(mod\ N). \tag{5}$$

The encrypted message is fairly secure due to hardship of integer factorization with large primes. Naturally, the larger the prime numbers, the more difficult the factorization problem becomes. In real systems, very large numbers are used, typically

512-bit in size. In this project, integers of type *unsigned long long int* are used for the C++ implementation, which have a range of 64-bits.

In this implementation we convert plain text messages into their integer equivalences by firstly, converting a string into a character array using the *cin* member function *getline()* and also *strcpy_s()* form the string stream library, and secondly, converting each character to its ASCII equivalent integer [4]. The message is then encrypted into a ciphertext of integers. The reverse operation is used to return the ciphertext to its original message.

In option 4 of the program, the user can encrypt a text file. Each character of each line of the file is stored inside a vector of vectors (see line 274 of main.cpp). An RSA object array is created for each line, meaning each character is encrypted using different keys, this increases the effectiveness of the encryption, making it harder to resolve. A separate file is generated along with the encrypted file containing the public keys. When a file is to be decrypted, the public key file is read alongside the cipher file. See Fig 4.


mess.txt - Notepad
File Edit Format View Help
cipher.txt - Notepad
File Edit Format View Help
public_key.txt - Notepad
File Edit Format View Help

```
this is a message
this is a message
this is a message
this is amessage
this is amessagethis is amessage
```

```
8547 3211 28715 73748 29 64360 4600 122069 8041
97230 66968 57587 9598 32768 8108 59798 126474
92549 10052 6041 120172 14257 37685 3910 32768
2463 151238 6790 58895 194170 35004 75253 44837
14030 568143 100188 33339 43662 184794 5280 832
```

```
3,25912/26311 7,14040/14363 5,214368/215333 3,
5,185856/186839 5,73884/74429 7,167640/168479
7,95040/95677 3,30616/30967 7,39000/39803 5,24
5,3696/3827 3,161392/162271 3,32500/32881 7,10
5,31752/32551 13,595980/597557 3,116800/117493
```

Fig. 4. Input message file and output encrypted files containing ciphertext and public keys.

V. ALGORITHMS

A. *ADDITION CHAINING*

In RSA key generation, we want to calculate the modulo of an integer raised to an exponent. In general, this can be a large number due to the large prime numbers used. To increase speed and efficiency of the program it is necessary to take advantage of the distributive nature of exponentiation. Here, we use an addition chaining algorithm, as seen in Schneier, Bruce (1996), page 255. [1] This is a recursive operation involving smaller modulo reductions. In this project, this algorithm is used within the encryption and decryption member functions (beginning at lines 208 and 234 of Project.cpp).

B. *PRIMALITY*

The single parameter constructor in the RSA class generates random numbers using the *rand()* function, defined in the standard library. Use of the *rand()* function requires an initial random seed, implemented by the command,

```
srand (time(NULL));
```

The random number is then checked for primality. This is achieved by using Fermat's Little Theorum,

$$a^p \equiv a(mod\ p), \tag{6}$$

where p is a prime number and a is a natural number. Implementation of this primality check can be found starting at line 159 of Project.cpp. If condition (3) does not hold then the initial random number becomes itself minus one, and the check is repeated until primality is found.

C. *EXTENDED EUCLIDEAN ALGORITHM*

The chosen encryption exponent e in the RSA public key is required to share a greatest common divisor of unity with the Euler totient function ϕ. In the *gcd(int a, int b)* member function we use the extended Euclidean algorithm to achieve this (see line 70 of Project.cpp). The algorithm finds the solutions of,

$$ax + by = gcd(a, b), \tag{7}$$

where x and y are coefficients and a and b are integers. *gcd(a,b)* returns the greatest common divisor of a and b. Here, we use a tuple object to store the coefficients and input variable. Member functions of the tuple class; *get()*, *tie()* and *make_tuple()* are used to manipulate and retrieve the variables.

VI. DISCUSSION

In this project, a class hierarchy is devised to support multiple cryptosystems. Although implementation of only the RSA algorithm is presented here, future development would involve integrating a selection of systems of different types. One area of cryptology that is very interesting and is used in modern day is elliptic curve cryptography. Future work on this project would try to implement such types to give the user a choice of algorithm to be used for encryption. This project involves basic techniques of C++ object orientated programming. A reader interested in a more advanced and complete set of cryptographic algorithms implemented in C++ should look at the open-source class library Crypto++. [5]

REFERENCES

[1] Schneier, Bruce (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition (2nd ed.). Wiley. ISBN 978-0-471-11709-4.

[2] Galbraith, S. (2012). Mathematics of public key cryptography. Cambridge: Cambridge Univ. Press.

[3] HOFFSTEIN, J. (2016). Introduction to mathematical cryptography. SPRINGER-VERLAG NEW YORK.

[4] En.cppreference.com. (2019). ASCII Chart - cppreference.com. [online] Available at: https://en.cppreference.com/w/cpp/language/ascii [Accessed 1 May 2019].

[5] Cryptopp.com. (2019). Crypto++ Library 8.2 | Free C++ Class Library of Cryptographic Schemes. [online] Available at: https://www.cryptopp.com/ [Accessed 1 May 2019].